BLOCKMAKER LIVE

Guía de ejercicios

Módulo 7

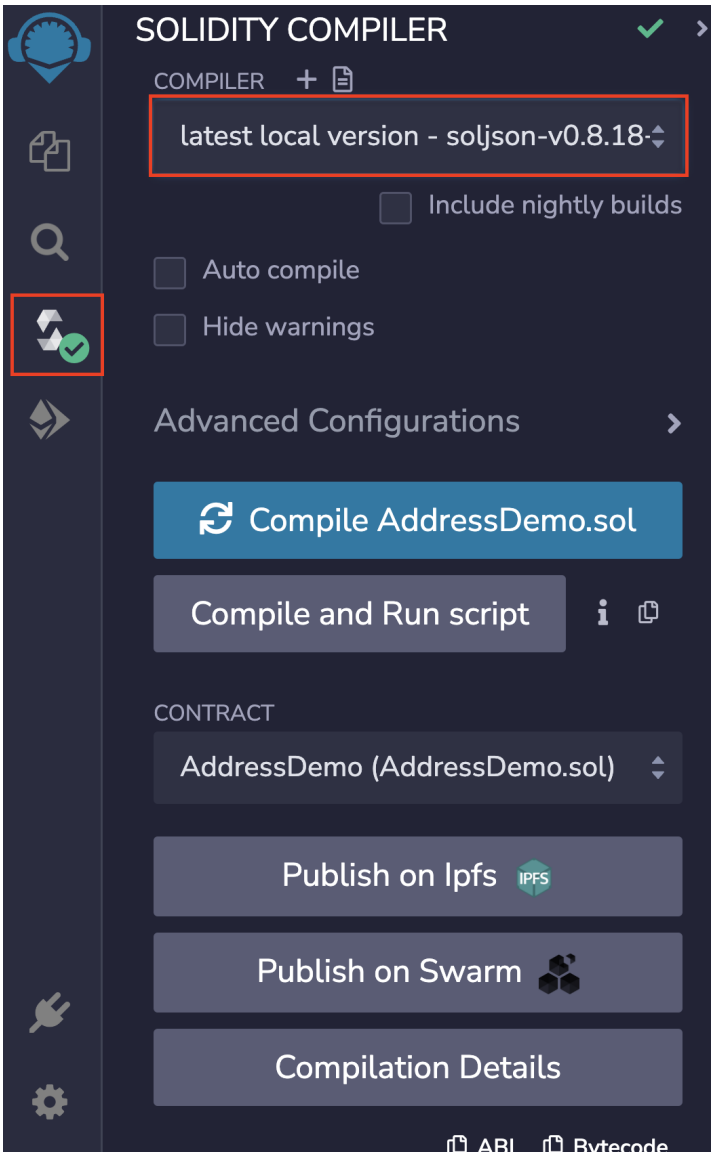
Recordatorios

1.A continuación, os dejamos un enlace a modo de bibliografía y de utilidad para consultar cualquier duda en materia de Solidity (si lo necesitáis, usar la función de traducir página al español):

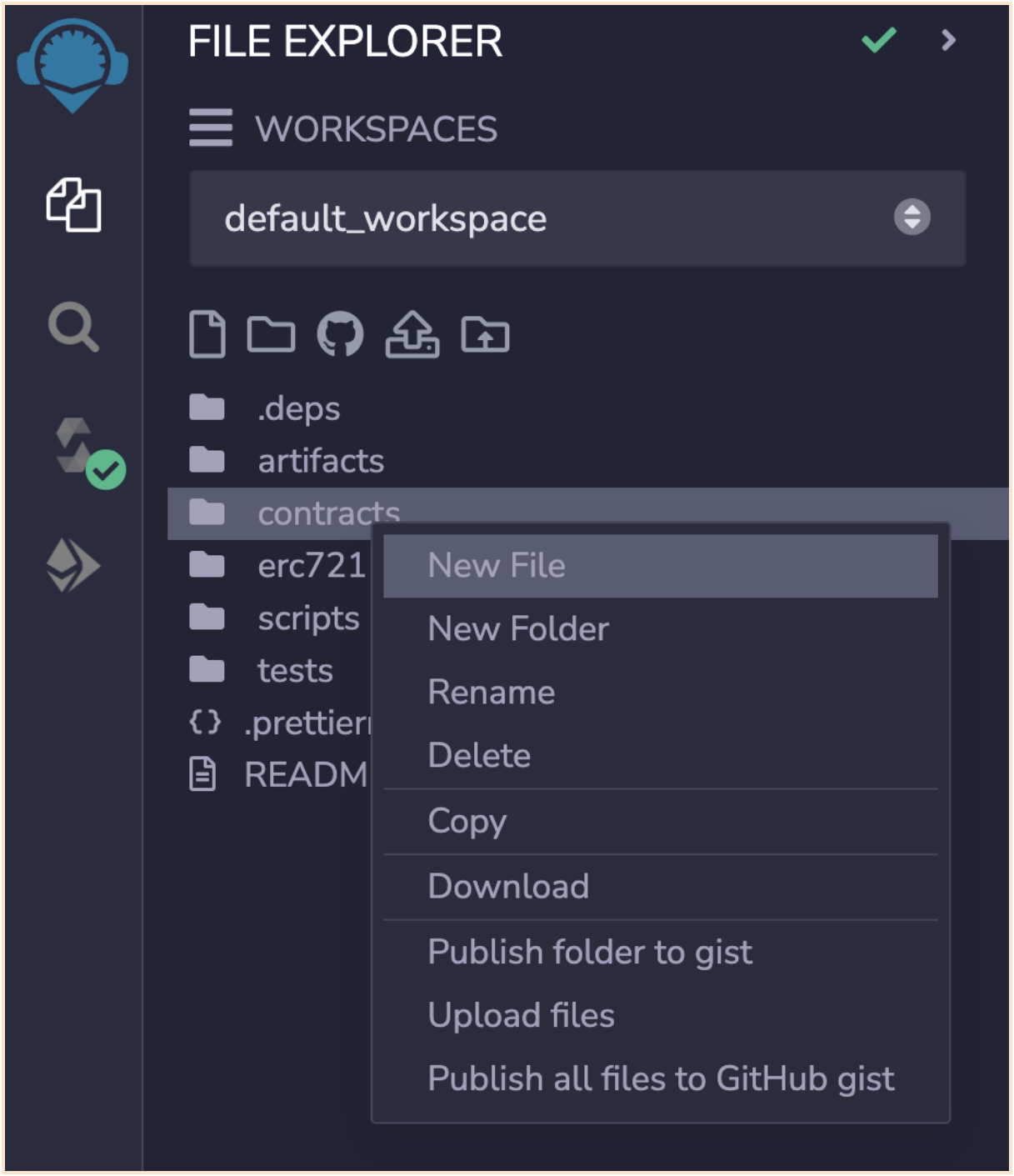
- https://docs.soliditylang.org/en/v0.8.21/grammar.html

2.Para realizar estos ejercicios vamos a utilizar el compilador online de Remix http://remix.ethereum.org/

RECUERDA 1: Desde la página http://remix.ethereum.org/ vamos a irnos a la tercera sección de la izquierda y vamos a cambiar la versión del compilador. Para ello, seleccionaremos cualquiera superior a la 0.8.0:



RECUERDA 2: Para crear un nuevo contrato, nos iremos a la primera sección de la izquierda y colocaremos en la carpeta “contracts” -> “New File” y le pondremos una nombre con la extensión “.sol.



Paso 0. Crear el contrato y descripción general

Crea un nuevo contrato Solidity con el nombre "**Subasta**."

Descripción del contrato: La idea general del siguiente contrato de subasta es que todo el mundo puede enviar sus pujas durante un periodo estipulado de puja.

- Las pujas ya incluyen el envío de alguna compensación, por ejemplo Ether, con el fin de vincular a los pujadores a su puja.
- Si se sube la puja más alta, el anterior mejor postor recupera su Ether. Una vez finalizado el periodo de pujas, el contrato tiene que ser llamado manualmente para que el beneficiario reciba su Ether - los contratos no pueden activarse por sí mismos.

Paso 1. Introducir cabecera contracto

Copiar al inicio del contrato inteligente.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
```

Paso 2. Declarar “contract” Subasta

Creamos el **contract Subasta** (recuerda que se declara como si fuera una clase y se le añaden las llaves para introducir dentro todo el contenido).

Paso 3. Crear variable propietario

Define una **variable de estado llamada "propietario"** de tipo **address** y haz que sea **pública (public)**. Esta variable almacenará la dirección del propietario del contrato y de la subasta.

Paso 4. Crear variable beneficiario

Define una **variable de estado llamada "beneficiario"** de tipo **address payable** y haz que sea **pública (public)**. Esta variable almacenará la dirección del beneficiario de la subasta.

Paso 5. Crear variable finSubasta

Define una **variable de estado llamada "finSubasta"** de tipo **entero** y haz que sea **pública (public)**. Esta variable almacenará el momento de finalización de la subasta.

Paso 6. Crear variable mejorPostor

Define una **variable de estado llamada "mejorPostor"** de tipo **address** y haz que sea **pública (public)**. Esta variable almacenará el address de la persona con la puja más alta.

Paso 7. Crear variable mejorOferta

Define una **variable de estado llamada "mejorOferta"** de tipo **entero** y haz que sea **pública (public)**. Esta variable almacenará la cantidad de la puja más alta.

Paso 8. Crear variable devolucionesPendientes

Define una **variable de estado llamada "devolucionesPendientes"** de tipo **mapping** que relacione **address** con un **entero**. Esta variable almacenará el address de todos aquellos postores cuya puja ya no se encuentra vigente debido a alguna otra. El número entero representa la cantidad que se le debe devolver al propio postor (ya que ha enviado ether al contrato para poder realizar la puja y ahora debemos devolvérselo).

Paso 9. Crear variable finalizada

Define una **variable de estado llamada "finalizada"** de tipo **boolean** que reflejará si la subasta ha terminado o no.

Paso 10.Crear evento OfertaMasAltaIncrementada

Crea un **evento (event)** llamado " **OfertaMasAltaIncrementada** " con los siguientes datos:

- postor (address)
- cantidad (entero)

Evento para registrar el establecimiento de una nueva puja máxima.

Paso 11. Crear evento SubastaFinalizada

Crea un **evento (event)** llamado " **SubastaFinalizada**" con los siguientes datos:

- ganador (address)
- cantidad (entero)

Evento para registrar al ganador definitivo de la subasta.

Paso 12. Crear errores

Crea cuatro errores bajo la siguiente nomenclatura:

```
/// La subasta ya ha finalizado.
error SubastaYaFinalizada();
/// Ya existe una oferta igual o superior.
error OfertaNoSuficientementeAlta(uint256 mejorOferta);
/// La subasta aún no ha finalizado.
error SubastaNoFinalizadaTodavia();
/// La función auctionEnd ya ha sido llamada.
error FinSubastaYaLlamado();
```

Dichos errores serán lanzados antes determinadas circunstancias junto a un “revert” (lo veremos en la clase pertinente).

Paso 13. Crear modificador soloPropietario

Crea un **modificador (modifier)** llamado " **soloPropietario**" con la siguiente condición:

- Comprobamos que el emisor de la petición sea el propietario y sino devolvemos un mensaje de error.

Paso 14. Crear constructor

Crea una **función constructora (constructor)** que contenga:

***Parámetros de entrada**:*

- Variable **tiempoOferta** de tipo **entero**. A través de esta pasaremos el tiempo en segundos que durará la oferta (e.g., si fuera 10 minutos deberíamos pasar = 10 min * 60 seg = 600 seg).
- Variable **addressBeneficiario** de tipo **address payable**, que representará la persona a la que se transferirán los fondos una vez que la subasta finalice.

***Descripción**:*

- Igualamos la variable **beneficiario** a la variable local / entrada de función addressBeneficiario**.**
- Igualamos la variable **finSubasta** al momento actual a nivel de fecha y tiempo (block.timestamp) + la cantidad de segundos que nos han enviado bajo la variable tiempoOferta.

Paso 15. Implementar función ofertar

Implementar la función "**ofertar**" **pública** y **payable** que permitirá establecer una puja en la subasta.

***Descripción**:*

- Comprobamos (mediante un if) que la subasta no ha terminado. Para ello, cogeremos el momento actual a nivel de fecha y tiempo (**block.timestamp**) veremos si es superior a la fecha de **finSubasta**.
    - En caso de que el periodo de la subasta ya haya terminado, lanzaremos un **revert** con el error **SubastaYaFinalizada()**.
- Tras esto, comprobamos (mediante un if) si el valor recibido en ether es inferior o igual a la actual **mejorOferta**.
    - En caso de que la cantidad de ether enviada sea igual o menor, lanzaremos un **revert** con el error **OfertaNoSuficientementeAlta()**.
- Finalmente, comprobamos (mediante un if) si la mejor oferta que nos han enviado es distinta de 0.
    - En caso de así sea, sumamos al mapping de **devolucionesPendientes** del **mejorPostor** actual la **mejorOferta** que realizó, ya que ha sido sobrepujado y habrá que devolverle la puja inicial.
- Una vez hemos realizado todas las comprobaciones, estableceremos la variable **mejorPostor** con el valor del nuevo ganador y la variable **mejorOferta** con el valor en ether que ha enviado el emisor.
- Finalmente, emitimos el evento **OfertaMasAltaIncrementada** con el emisor y la cantidad de ether enviado.

Paso 16. Implementar función retirar

Implementar la función "**retirar**" **pública** que permitirá retirar al usuario que previamente ha pujado, pero ya no es el ganador, la cantidad de ether enviada inicialmente.

Parámetros de salida:

- **booleano** que devuelva true si se ha retirado la cantidad.

Descripción*:*

- Inicialmente, crearemos una variable local llamada **cantidad** de tipo **entero** que sea igual a la cantidad pendiente de retirar por el emisor (recordar el mapping **devolucionesPendientes**).
- Comprobamos (mediante un if) si la **cantidad** es superior a 0.
    - En caso afirmativo, dejamos a 0 la cantidad pendiente por devolver al emisor (**devolucionesPendientes**).
    - Comprobamos (mediante un if) si no conseguimos realizar el **<address_payable>.send(cantidad)** al emisor de la petición de retirar.
        - En caso de que esto sea false, es decir, que no consigamos enviar la cantidad pertinente mediante al función send, establecemos de nuevo **devolucionesPendientes** del emisor igual a la **cantidad** previa.
        - Tras esto, **retornamos false**.
- Finalmente, **retornamos true** al completar el proceso.

Paso 17. Implementar función finalizarSubasta

Implementar la función "**finalizarSubasta**" **pública** que permitirá poner fin al proceso de subasta y establecer el ganador final.

Modificador:

- **soloPropietario**.

Descripción*:*

- Inicialmente, comprobaremos (mediante if) si el tiempo de **finSubasta** establecido es mayor que la fecha y tiempo actual.
    - En caso de que así sea, significará que la subasta aún no ha llegado a la fecha límite, por lo que lanzamos un **revert** con el error **SubastaNoFinalizadaTodavia()**.
- Comprobamos (mediante if), si la variable **finalizada** está a true o false.
    - En caso de estar a true, significaría que la subasta ya ha finalizado y lanzaríamos un **revert** con el error **FinSubastaYaLlamado()**.
- Tras estas comprobaciones y sabiendo que la subasta ya puede finalizarse, establecemos la variable **finalizada** a **true**.
- Emitimos el evento **SubastaFinalizada** pasándole el **mejorPostor** y la **mejorOferta**.
- Finalmente, transferimos los fondos de la subasta al **beneficiario** que habíamos establecido previamente (en esta ocasión utilizaremos **transfer** ya que no queremos controlar los posibles errores por nuestra cuenta).